BLM3021 - Algoritma Analizi
Ödev 1: Problem 1

Mesut Şafak Bilici
17011086
l1117086@std.yildiz.edu.tr

07/11/2020

(a) Ödev Konusu

Ödevin (a) şıkkında, verilen problemi Brute-Force bir yaklaşımla çözen algoritmayı yazmamız ve bunu analiz etmemiz isteniyor. Bu problem ise N elemanlı bir dizide birbirine en yakın değere sahip iki elemanı bulmak. Daha sonra verilen bu problemin verimliliğini analiz etmemiz isteniyor.

(a) Algoritma

(a) şıkkında bizden Brute-Force bir yaklaşım istendiği için olabildiğince sade bir çözüm üretilmiştir. Çözüm iç içe iki for içermektedir. A[0..$n-1$] dizisi ve i iteratörü için dışardaki for döngüsü 1. elemandan $n-2$. elemana kadar (dahil) gitmektedir. A[0..$n-1$] dizisi ve j iteratörü için içerdeki for döngüsü ise $i+1$. elemandan $n-1$. elemana kadar (dahil) gitmektedir. İçerde ise her A[i] ve A[j] ikilisi arasındaki farkın mutlak değerine bakılıyor. Eğer bu mutlak değer farkı bir önceki en küçük değerden küçükse, yeni en yakın değer olarak min değişkenine atanıyor.
Bu işlemler `int minDiff(int*,int,int*,int*)` fonksiyonu içinde yapılıyor. İlk `int*` parametresi array'imiz iken, ikinci `int` parametresi array'imizin boyutu oluyor. Bizden en yaklaşık değer ve elemanlar istendiği için fonksiyon değeri döndürüyor, elemanları da alabilmek için `int*,int*` şeklinde iki farklı indis parametresi aktarılıyor.

(a) Verimlilik Analizi

Array'i bastırmak ve array'i dinamik olarak oluşturmak tek for içinde olduğu için bizim analiz etmemiz gereken nokta iki for'a sahip olan asıl minimum'u bulma algoritması. For döngüsünün içinde her adımda yapılan 2 işlem var : aralarındaki uzaklığı atama işlemi ve minimum mu değil mi karşılaştırması. Bu yüzden running time'ı şu şekilde yazabiliriz ($A_{\text{worse}}(n)$: atama, $C_{\text{worst}}(n)$: karşılaştırma):

$$T_{\text{worst}}(n) = A_{\text{worst}}(n) + C_{\text{worst}}(n) \tag{1}$$

$$A_{\text{worst}}(n) = \overbrace{\sum_{i=0}^{n-2} \underbrace{\sum_{j=i+1}^{n-1} 1}_{\text{inner for}}}^{\text{outer for}} \quad \text{ve} \quad C_{\text{worst}}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \tag{2}$$

$$C_{\text{worst}}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} \left(\sum_{j=i+1}^{n-1} 1 \right) \tag{3}$$

$$= \sum_{i=0}^{n-2} (n-i-1) = \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \tag{4}$$

$$= \left((n-1) \cdot \sum_{i=0}^{n-2} 1 \right) - \frac{(n-2)(n-1)}{2} \tag{5}$$

$$= (n-1) \cdot (n-1) - \frac{(n-2)(n-1)}{2} \tag{6}$$

$$= \frac{n(n-1)}{2} \approx \frac{1}{n} n^2 \in \Theta(n^2) \tag{7}$$

Aynı şekilde $A_{\text{worst}}(n)$'in $\Theta(n^2)$ olduğu gösterilebilir. Bu durumda

$$T_{\text{worst}}(n) = A_{\text{worst}}(n) + C_{\text{worst}}(n) \in \Theta(n^2) + \Theta(n^2) \tag{8}$$

$$\in \Theta(n^2) \tag{9}$$

şeklinde algoritmik karmaşıklığımız ve verimliliğimiz gösterilebilir.

Çıktılar

```
/media/safak/Data/github/YTU-CE/3x5/BLM3021 - Algorithm Analysis/HMWs/HMW1 | master !1 ?3
❯ ./makefile1a
Enter the number of element of array: 10
Enter the element 0: 555
Enter the element 1: 999
Enter the element 2: 551
Enter the element 3: 645
Enter the element 4: 845
Enter the element 5: 315
Enter the element 6: 12
Enter the element 7: 111
Enter the element 8: 700
Enter the element 9: 42

Array: [ 555 999 551 645 845 315 12 111 700 42 ]
Minimum difference is between index 0 and 2 and equal to 4.%
```

Figure 1

```
/media/safak/Data/github/YTU-CE/3x5/BLM3021 - Algorithm Analysis/HMWs/HMW1 | master !1 ?3
❯ ./makefile1a
Enter the number of element of array: 5
Enter the element 0: 4
Enter the element 1: 9
Enter the element 2: 100
Enter the element 3: 101
Enter the element 4: 32

Array: [ 4 9 100 101 32 ]
Minimum difference is between index 2 and 3 and equal to 1.%
```

Figure 2

(b) Ödev Konusu

Ödevin (b) şıkkında, (a) şıkkında verilen problemi daha efektif (daha az karmaşıklığa sahip) bir yaklaşımla çözen algoritmayı yazmamız ve bunu analiz etmemiz isteniyor.

(b) Algoritma

(b) şıkkında bizden (a) şıkkına kıyasla daha efektif bir algoritma yazmamız istenmiştir. Brute-Force bir yaklaşım yerine efektif bir sıralama algoritması (algoritmam için merge sort) ile sıralayıp daha sonra A[i], A[$i+1$] ikilileri arasındaki uzaklıklara bakak yeterli olacaktır, ikili arasındaki uzaklık önceki bulunan en düşük uzaklıktan küçükse, yeni en düşük uzaklık olarak güncellenecektir. `void mergeSort(int*,int,int)` ve `void merge(int*,int,int,int)` merge sort için gerekli fonksiyonlarımız olmaka beraber, yukarda bahsedilen tek for içinde halledilecek ikililere bakma fonksiyonu `int minDiff(int*, int, int*)` olarak tanımlanıp sırasıyla array'i, array uzunluğunu ve ikililerin ilk index'inin atanacağı indexi parametre olarak alıyor .

(b) Verimlilik Analizi

Tek for'a sahip olup ikililer arasında uzaklık hesaplayıp minimum kontrolü yapan fonksiyonun karmaşıklığı $\Theta(n)$ olacaktır. Bunun dışında diziyi sıralamak için merge sort kullandım. Merge sort için karmaşıklığımız

$$C(n) = \begin{cases} 2C(n/2) + C_{\text{merge}}(n), & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases} \tag{10}$$

şeklinde ifade edilebilir. Master theorem için a,b ve d değerlerimiz sırasıyla 2,2 ve 1 olacaktır. Bu şekilde karmaşıklığımız $\Theta(n^d \log n) = \Theta(n \log n)$ olur. Algoritmamızın genel karmaşıklığı da $\Theta(n \log n)$ olup Brute-Force yöntemden bariz bir şekilde daha verimlidir.

Çıktılar

```
/media/s/D/g/YTU-CE/3x5/BLM302/H/HMW1 | master !1 ?3
❯ ./makefile1b
Enter the number of element of array: 9
Enter the element 0: 33
Enter the element 1: 36
Enter the element 2: 99
Enter the element 3: 78
Enter the element 4: 54
Enter the element 5: 1
Enter the element 6: 845
Enter the element 7: 621
Enter the element 8: 129

Array: [ 33 36 99 78 54 1 845 621 129 ]

Sorted Array: [ 1 33 36 54 78 99 129 621 845 ]
Minimum difference is between index 1 and 2 and equal to 3.%
```

Figure 3

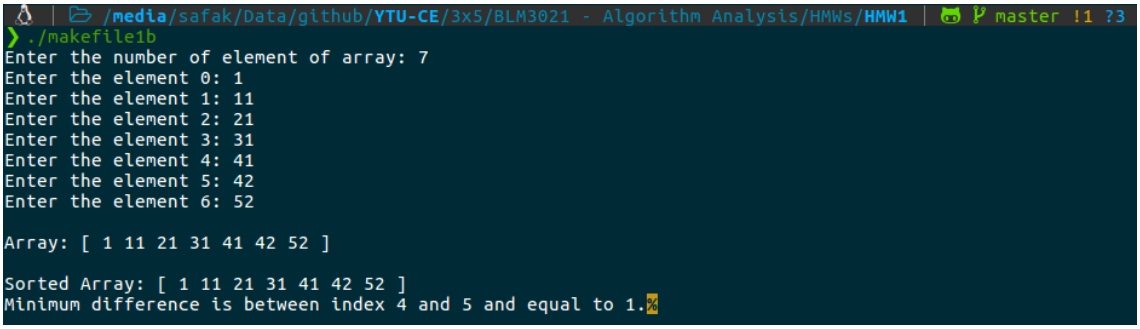

Figure 4

Kodlar

Sırasıyla 17011086_1_a.c ve 17011086_1_b.c